Investigating Applicability Heuristics of Answer Set Programming in Game Development: Use Cases and Empirical Study

Evangelos Lamprou
e.lamprou@upnet.gr
University of Patras
Patras, Greece

Christos Fidas
fidas@upatras.gr
University of Patras
Patras, Greece

ABSTRACT

The game industry is continuously growing and evolving, with new ways of creating games being developed. However, even with the availability of powerful game engines, developers are still forced to spend time and effort implementing common game features, such as basic AI, path-finding, and simple scene variations. This can become a serious detriment for indie game developers. The present research focuses on the application of Answer Set Programming (ASP) methods within the game development process, aiming to support rapid and cost-effective game prototyping for indie game developers. Specifically, we present a pragmatic approach to the usage of ASP for game development within certain use cases and we report on evaluation results based on feedback that was received from end-users. Analysis of results demonstrates how ASP can be used, providing new ways of thinking about game mechanics and content creation, and eventually paving the way for new game design frameworks and possibilities. On the downside, adoption of the suggested method can be difficult due to unfamiliarity with the ASP programming paradigm.

CCS CONCEPTS

• **Human-centered computing** → *Empirical studies in HCI*; • **Computing methodologies** → **Logic programming and answer set programming**; • **Software and its engineering** → *Software prototyping*; *Interactive games*.

KEYWORDS

Game Development Answer Set Programming Evaluation Study

ACM Reference Format:
Evangelos Lamprou and Christos Fidas. 2023. Investigating Applicability Heuristics of Answer Set Programming in Game Development: Use Cases and Empirical Study . In *2nd International Conference of the ACM Greek SIGCHI Chapter (CHIGREECE 2023), September 27–28, 2023, Athens, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3609987.3609993

1 INTRODUCTION

The game industry continuously evolves, and new ways of creating games are emerging. Game engines offer powerful tools and features to aid game designers in realizing their ideas [1]. However, most game development engines rely on imperative languages, which specify step-by-step instructions using variables, loops, and conditional statements. In contrast, Answer Set Programming (ASP) [12] is a declarative programming paradigm that has shown promise in solving complex problems across various fields, including games [5, 6, 16, 23]. ASP has been applied in puzzles, game playing, procedural content generation, and game content generation tools [3, 4, 8, 10, 13, 25, 28, 29].

The exploration of different programming languages and paradigms in game development has been a subject of study [2, 20, 24, 26]. However, there is still a gap in understanding how to effectively apply these innovative tools as an end user [11].

Motivation and Contribution. In small game development teams (1-5 people), the roles of developer and designer often overlap. This means game designers frequently interrupt the creative process to implement complex game logic. However, there is a lack of suitable tools for rapid prototyping. To address this issue, we propose an ASP framework for game development. We highlight aspects of game development suitable for ASP implementation, present case studies, and evaluate ASP with developers of varying familiarity with the paradigm.

The paper is structured as follows. First we present background knowledge for understanding the ASP programming paradigm. Next, we present the suggested framework that can be utilized by indie game developers and finally, we present the results of the evaluation study.

2 BACKGROUND THEORY ON ANSWER SET PROGRAMMING

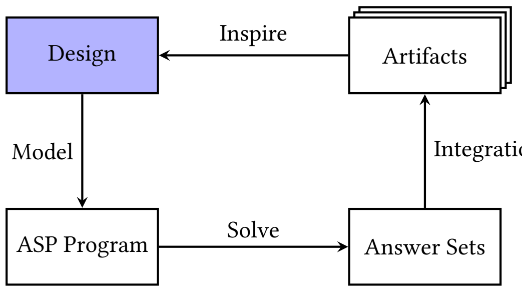

Figure 1: Game development workflow with the help of ASP tooling. The designer starts with an initial goal, the design of a game mechanic/behavior/set of artifacts which leads to a specification in the form of an ASP program. The program's solutions can help to further refine the initial design as missing or unwanted aspects of it become apparent after its integration with the rest of the game [24].

Answer Set Programming (ASP) [12, 18] is a problem-solving paradigm with roots in logic programming and non-monotonic reasoning. As shown in fig. 1, the programming model of ASP is one where the programmer models the problem domain, with the solution being handled by a solver program. Programming using this paradigm is done in a family of languages sometimes called *AnsProlog* [17]. In our work, we will be using the input language of *Clingo* [15], which is a high-performance integrated solver with a large collection of libraries and bindings helpful in integrating it with external tools.

The syntax, similar to that of *Prolog*, represents code and data through logical terms. Collections of logical terms, as seen in listing 1, can represent the game world's state. Rules, expressed using the `:-` operator, enable complex reasoning by defining relationships between atoms. Choice rules, exemplified in listing 3, allow the ASP solver to make selections among atoms. Integrity constraints, as shown in listing 4, can restrict invalid answer sets. Optimization directives, such as `#minimize` and `#maximize` in listing 5, guide the solver to output optimal answer sets.

```
position(player, vec3(1, 0, 1)).
tile(1, 1, water).
object(orc).
object(frog).
```

Listing 1: A set of facts describing game elements/game state.

```
damaged(player) :- fall(player).
hostile(X) :- enemy(X).
friend(X) :- object(X), not hostile(X).
health(N-1, T+1) :- health(N, T), attacked(T).
```

Listing 2: Logic rules describing relationships between entities and connection between action and effect.

```
{chosen(X,Y) : person(X)} :- house(Y).
```

Listing 3: A choice rule.

```
:- chosen(X, Y), chosen(Z, Y), X == Y.
```

Listing 4: An integrity constraint.

```
#minimize{C : cost(E,C)}.
```

Listing 5: An optimization directive.

To compute answer sets, ASP programs are inputted into ASP solvers. These solvers provide efficient mechanisms to generate the set of valid answers to the given problem. An ASP solver can be thought of as a black box, with them being interchangeable as long as the input language semantics remain the same.

3 HEURISTICS AND METHOD FOR APPLYING ASP IN THE GAME DEVELOPMENT PROCESS

An important aspect of the suggested framework is the determination of specific game design heuristics pointing towards game components suitable for ASP programming approaches. The set of heuristics was constructed by identifying common features from games developed using ASP in the literature while also considering the paradigm's technical limitations in the context of games. We suggest the following applicability principles/heuristics:

- ***ASP Applicability Heuristic (A).*** **Brevity**: ASP (and declarative programming in general) can reduce software complexity [27], allowing for concise code [7]. This however requires that when modelling a game mechanic, only its important aspects are encoded. For example, in a maze game, only essential elements like maze layout, starting point, treasure location, and movement rules would be included.
- ***ASP Applicability Heuristic (B).*** **Relatively Small Solution Space**: To minimize solving times, designers should avoid scenarios with large solution spaces. One way to achieve this is by limiting the options available to the generator/agent through choice rules in the ASP program. For instance, restricting an agent's movement to four directions (up, down, left, right) instead of full motion. To introduce natural-looking movement later, a physics simulation can be utilized within the game engine.
Designers can further address this limitation by breaking down the problem into smaller sub-problems. For example, in [9], the generation of a dungeon's topology was decoupled from the content of each room, reducing generation times. In [22], an agent's different states (eat, hiding, action) were split into smaller ASP program modules, leveraging meta-reasoning to determine the relevant parts of the knowledge base for solving.
- ***ASP Applicability Heuristic (C).*** **Emergent Complexity**: Create scenarios where interesting behavior emerges when agents are observed interacting with each other and the environment inside the game world or when the generated artifacts exhibit interesting patterns that were not explicitly modeled in the ASP program. In [21], where a declarative planning layer was added to agents in the game *F.E.A.R*, complex behaviors emerged from a combination of simple goals and actions together with the dynamic state of the game world. The *Portal* game levels generated using ASP in [4] were complex and challenging, while modeling only involved specifying how a level is solvable and ensuring its topological integrity.

Furthermore, we propose a standardized development methodology (fig. 2) to guide aspiring developers to successfully apply ASP to their applications by providing some general programming guidelines relating to ASP modelling, based on the "guess and check" paradigm [14].

1. **Step (a): Determine Input and Output Atoms**: The set of input atoms provides the context required for the ASP program to give correct results. These are usually dynamic aspects of the game's runtime and change at each invocation of the ASP solver. On the other hand, output atoms encode the results produced by the solver and which will be interpreted by the game runtime as artifacts or agent behavior.
2. **Step (b): Generate "Random" Answer Sets**: During the development phase, the programmer can efficiently construct an ASP program comprising of choice rules to generate partially random outcomes, considering the output atoms. This stage also involves the integration of the solver with the game runtime to enhance the debugging process.

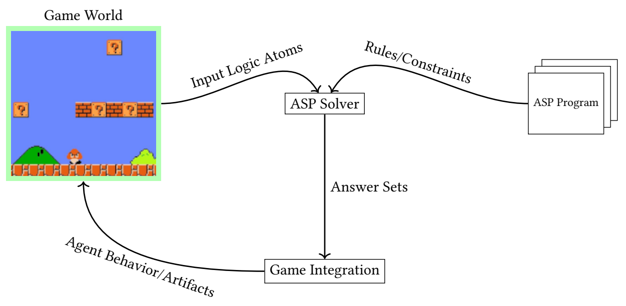

Figure 2: High level overview of ASP integration into a game engine. The *Game World* consists of the current game state and the information of all the game objects inside of it. Input logic atoms are the facts used to describe the current state of the game world. These, together with the rules and constraints of the ASP program, are fed into the ASP solver, which then outputs answer sets. These describe logic such as the actions that an agent should take or where an object should be placed. Through the *Game Integration* component, these answer sets are used to update the game world.

(3) **Step (c): Add Integrity Constraints/Optimization Directives**: Based on the current problem's domain, it is necessary to add integrity constraints and/or optimization directives. Constraints provide direct control over the produced answer sets for them to comply both with the game's ruleset as well as the designer's ideas. Among them, if needed, the solver can output the most optimal ones based on some variable using optimization directives.

3.1 Proof of Concept and Use Cases

Proof of concept applications were developed to demonstrate the practical implementation of ASP in games. These applications serve as concrete examples, showcasing how ASP can be utilized to implement game mechanics and content generators and have been made publicly available [19].

4 EMPIRICAL STUDY

4.1 Research Questions

The main research questions of the empirical study were to investigate: **RQ1**) Can the aforementioned applicability heuristics be validated by beginner game developers within the context of their own game designs? **RQ2**) What does the proposed methodology offer specifically to beginner game developers? **RQ3**) Did beginner game developers encounter challenges when applying the suggested workflow, and if so, what were the specific difficulties faced?

4.2 Participants

We recruited a total of 8 participants (2 female and 6 male), all of whom were undergraduate *Electrical and Computer Engineering* students. All of them had experience programming with imperative languages, with 3 having experience with logical languages (either *Prolog* or *Clingo*). All but two of the participants had prior experience with game development, in the context of personal projects. Participants were informed that no personal data was collected aside from their answers to the interview part of the study. On average, each of the participants took part in the study procedure for a duration of 0.5 to 3 hours, resulting in a total study length of approximately twenty-four hours.

4.3 Study Procedure

The study conducted was a one-on-one user study, where each participant worked individually with the researcher. The study utilized all collected data in an anonymous manner, and participants had the freedom to withdraw from the study at any time of their choosing.

- **Phase A - Introduction to ASP.** The study began with a brief overview of Answer Set Programming (ASP) technology and the *Clingo* language's syntax and semantics.
- **Phase B - Implementation of Game Mechanic/Generation of Content.** Study members took the role of a non-expert game developer who is tasked both with designing and implementing part of a game, a scenario present in indie game development contexts. Participants were encouraged to think of a game mechanic or content generator to implement using ASP, fostering creativity and challenging ideas. The researcher assisted participants in creating the logic program for their game mechanic with ASP, providing guidance and answering questions. Despite the mature programming background of the participants, we avoided the use of a traditional imperative approach as part of the experiment, which, even though would have provided a concrete baseline for comparison, could potentially limit participants' design choices based on familiarity with imperative methods.
- **Phase C - Discussion.** Finally, we conducted a semi-structured interview to receive qualitative feedback and elicit the participant's likeability and comments concerning the proposed workflow.

Limitations. Certainly, a limitation of our study is that the participants' profile was limited to students rather than experienced game developers. Additionally, the number of participants was relatively small, resulting in our reliance on qualitative analysis for the research findings.

5 ANALYSIS OF RESULTS

5.1 RQ1: Applicability of suggested ASP Heuristics in Game Development Scenarios

During the study, a range of applications were created [19]. Participants demonstrated competence in applying the ASP applicability heuristics throughout the study. Their creations showcased an understanding of at least one of brevity, consideration of relatively small solution spaces, and the ability to capture emergent complexity. These observations affirm the practicality and effectiveness of the ASP applicability heuristics in guiding participants towards successful implementation and utilization of ASP in the game development process.

5.2 RQ2: Value of the Methodology

A participant with extensive game development experience stated that the workflow provides the opportunity to conceive entirely novel game mechanics that would otherwise be overlooked due to the challenges associated with traditional programming. This participant emphasized that Answer Set Programming (ASP) allows for the creation of game mechanics that might otherwise be deemed too complex to develop.

The majority of participants valued ASP primarily for its concise problem-solving capability, streamlining development. It was noted that ASP can benefit non-programmers by enabling logical constraint expression without extensive programming experience. After introduction to the ASP paradigm, most participants easily recognized applicable scenarios. Among the proposed design heuristics, the "relatively small solution space" was the most challenging to apply and perceived as limiting by participants. Despite a mention of a steep learning curve by some, the ASP approach was generally regarded as a powerful and creative method for game design.

- *Participant 1*: "It gives you the ability to create entirely new game mechanics that you wouldn't bother developing otherwise because of the programming difficulty."
- *Participant 5*: "[A game developer] might say something like "Oh, this can be easily encoded using rules". Now, it's easier to think of a game mechanic and come up with constraints to create it."

5.3 RQ3: Difficulties in the suggested ASP workflow application

We asked participants what they found challenging about applying the proposed workflow. Participants encountered challenges with the unfamiliar syntax in *Clingo*, finding it unconventional and difficult to write and read programs. The lack of a debugger hindered error identification, especially during complex iterations with multiple rules and constraints. Additionally, slow solving times and limited scalability were recognized as limitations.

- *Participant 1*: "I believe it would take me a long time to learn the language. The syntax is strange, but I can create a mental model of how it works."
- *Participant 7*: "It would be nice to have a graphical interface that shows how the solver arrives at solutions."
- *Participant 8*: "[The workflow] could be improved if Clingo had better syntax. Maybe an abstraction layer built on top of it."

6 CONCLUSIONS AND FUTURE WORK

The aim of our research was to explore the applicability heuristics of Answer Set Programming (ASP) in the context of game development. We focused on identifying potential use cases where ASP could be effectively utilized and conducted an empirical study to evaluate its practical effectiveness. To achieve our research objectives, we first identified potential use cases where ASP could be applied. Furthermore, we conducted an empirical study to assess the practical feasibility and effectiveness of using ASP in game development. This study involved designing experiments or scenarios where ASP-based solutions were implemented and evaluated. The proposed workflow stems from the need for robust high level programming interfaces to assist with the development of complex applications like games. We have proposed a methodology for recognizing parts where game logic can be elegantly expressed using answer-set programming. Through this, aspiring game developers, even without extensive experiencing with game creation, can incorporate this new set of tools to their workflow, enabling for a different approach and mental process for designing and implementing parts of their projects.

A research project that could overcome *ASP*'s narrow adaptation is the creation of language that preservers the *AnsProlog* language's semantics while providing a more developer-friendly syntax and structure. Future work should involve the application of ASP in larger-scale game projects, exploring how the proposed workflow can fit into long-running game projects.

ACKNOWLEDGMENTS

We would like to thank all the anonymous reviewers for their valuable input on this work. This research has been co-financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH – CREATE – INNOVATE (project code: T1EDK-T2EDK-01392).

REFERENCES

[1] A. Andrade. 2015. Game engines: a survey. *EAI Endorsed Transactions on Game-Based Learning Endorsed Transactions on Game-Based Learning* 2, 6 (nov 2015), 150615. https://doi.org/10.4108/eai.5-11-2015.150615

[2] Denise Angilica, Giovambattista Ianni, Francesca A. Lisi, and Luca Pulina. 2022. AI and Videogames: a "Drosophila" for Declarative Methods. In *Proceedings of the 10th Italian workshop on Planning and Scheduling (IPS 2022), RCRA Incontri E Confronti (RiCeRcA 2022), and the workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (SPIRIT 2022) co-located with 21st International Conference of the Italian Association for Artificial Intelligence (AIxIA 2022), November 28 - December 2, 2022, University of Udine, Udine, Italy (CEUR Workshop Proceedings, Vol. 3345)*, Riccardo De Benedictis, Nicola Gatti, Marco Maratea, Andrea Micheli, Aniello Murano, Enrico Scala, Luciano Serafini, Ivan Serina, Alessandro Umbrico, and Mauro Vallati (Eds.). CEUR-WS.org. https://ceur-ws.org/Vol-3345/paper8_RiCeRCa1.pdf

[3] Denise Angilica, Giovambattista Ianni, and Francesco Pacenza. 2022. Declarative AI design in Unity using Answer Set Programming. In *2022 IEEE Conference on Games (CoG)*. IEEE. https://doi.org/10.1109/cog51982.2022.9893603

[4] Evgenia Antonova. 2015. *Applying Answer Set Programming in Game Level Design.* Master's thesis. Aalto University.

[5] Theofanis Aravanis, Konstantinos Demiris, and Pavlos Peppas. 2018. Legal Reasoning in Answer Set Programming. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. https://doi.org/10.1109/ictai.2018.00055

[6] Georg Boenn, Martin Brain, Marina De Vos, and John ffitch. 2010. Automatic Music Composition using Answer Set Programming. arXiv:1006.4948 [cs.LO]

[7] Ivan Bratko. 2012. *Prolog programming for Artificial Intelligence.* Pearson education.

[8] Francesco Calimeri, Michael Fink, Stefano Germano, Andreas Humenberger, Giovambattista Ianni, Christoph Redl, Daria Stepanova, Andrea Tucci, and Anton Wimmer. 2016. Angry-HEX: An Artificial Player for Angry Birds Based on Declarative Knowledge Bases. *IEEE Trans. Comput. Intell. AI Games Transactions on Computational Intelligence and AI in Games* 8, 2 (jun 2016), 128–139. https://doi.org/10.1109/tciaig.2015.2509600

[9] Francesco Calimeri, Stefano Germano, Giovambattista Ianni, Francesco Pacenza, Armando Pezzimenti, and Andrea Tucci. 2018. Answer Set Programming for Declarative Content Specification: A Scalable Partitioning-Based Approach. (2018), 225–237. https://doi.org/10.1007/978-3-030-03840-3_17

[10] Kate Compton, Adam Smith, and Michael Mateas. 2012. Anza Island: Novel Gameplay Using ASP. In *Proceedings of the The Third Workshop on Procedural Content*

Generation in Games (Raleigh, NC, USA) *(PCG'12)*. Association for Computing Machinery, New York, NY, USA, 1–4. https://doi.org/10.1145/2538528.2538539

[11] Michael Debellis and Christine Haapala. 1995. User-centric Software Engineering. *IEEE Expert* 10 (03 1995), 34 – 41. https://doi.org/10.1109/64.391959

[12] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. 2009. Answer Set Programming: A Primer. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 40–110. https://doi.org/10.1007/978-3-642-03754-2_2

[13] D. Fuscà, Stefano Germano, J. Zangari, Francesco Calimeri, and Simona Perri. 2013. Answer set programming and declarative problem solving in game AIs. *CEUR Workshop Proceedings* 1107 (01 2013), 81–88.

[14] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. 2015. A User's Guide to gringo, clasp, clingo, and iclingo.

[15] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. 2014. Clingo = ASP + Control: Preliminary Report. *CoRR* abs/1405.3694 (2014). arXiv:1405.3694 http://arxiv.org/abs/1405.3694

[16] Martin Gebser, Roland Kaminski, and Torsten Schaub. 2011. aspcud: A Linux Package Configuration Tool Based on Answer Set Programming. *Electronic Proceedings in Theoretical Computer Science* 65 (aug 2011), 12–25. https://doi.org/10.4204/eptcs.65.2

[17] Michael Gelfond. 2002. Representing Knowledge in A-Prolog. In *Computational Logic: Logic Programming and Beyond*. Springer Berlin Heidelberg, 413–451. https://doi.org/10.1007/3-540-45632-5_16

[18] Michael Gelfond and Vladimir Lifschitz. 2000. The Stable Model Semantics For Logic Programming. *Logic Programming* 2 (12 2000).

[19] Evangelos Lamprou and Christos Fidas. 2023. *asp-games*. https://github.com/vagos/asp-games

[20] Chris Martens. 2021. Ceptre: A Language for Modeling Generative Interactive Systems. In *Artificial Intelligence and Interactive Digital Entertainment Conference*.

[21] Jeff Orkin. 2006. Three States and a Plan: The AI of F.E.A.R. GDC 2006.

[22] Tony Ribeiro, Katsumi Inoue, and Gauvain Bourgne. 2013. Combining Answer Set Programs for Adaptive and Reactive Reasoning. (dec 2013).

[23] Francesco Ricca, Giovanni Grasso, Mario Alviano, Marco Manna, Vincenzino Lio, Salvatore Iiritano, and Nicola Leone. 2011. Team-building with Answer Set Programming in the Gioia-Tauro Seaport. arXiv:1101.4554 [cs.LO]

[24] Adam M. Smith. 2012. *Mechanizing Exploratory Game Design*. Ph. D. Dissertation. University of California, Santa Cruz.

[25] Adam M. Smith and Michael Mateas. 2011. Answer Set Programming for Procedural Content Generation: A Design Space Approach. *IEEE Trans. Comput. Intell. AI Games Transactions on Computational Intelligence and AI in Games* 3, 3 (sep 2011), 187–200. https://doi.org/10.1109/tciaig.2011.2158545

[26] Adam M. Smith, Mark J. Nelson, and Michael Mateas. 2010. LUDOCORE: A logical game engine for modeling videogames. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE. https://doi.org/10.1109/itw.2010.5593368

[27] Diomidis Spinellis. 2013. The Importance of Being Declarative. *IEEE Software* 30, 1 (January/February 2013), 90–91. https://doi.org/10.1109/MS.2013.18

[28] Michael Thielscher. 2009. Answer Set Programming for Single-Player Games in General Game Playing. In *Logic Programming*. Springer Berlin Heidelberg, 327–341. https://doi.org/10.1007/978-3-642-02846-5_28

[29] Fernando Zacarias, Rosalba Cuapa, Luna Jimenez, and Noemi Vazquez. 2019. Modelling of Intelligent Agents Using A–Prolog. *International Journal of Artificial Intelligence and Applications* 10, 2 (mar 2019), 1–11. https://doi.org/10.4018/ijaia.2019030102